www.qconferences.com
www.qconbeijing.com
www.qconshanghai.com

伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

QCon全球软件开发大会

International Software Development Conference

www.infoq.com/cn

InfoQueue

@InfoQ

infoqchina

软件
正在改变世界！

Managing & Surviving Change

Ben Evans (ben@jclarity.com)

About Me

- **Java Champion & JavaOne Rock Star Speaker**
- **London Java Community Organising Team**
- **Author: Java程序员修炼之道 (Turing)**
- **Author: Java权威技术手册（第6版) - 2015年6月 (Turing)**
- **Java Community Process Executive Committee**
- **Founder & Tech Fellow, jClarity**
- **Previously:**
  - Chief Architect (Listed Derivatives) - Deutsche Bank
  - Morgan Stanley (MATRIX, Google IPO, Dodd-Frank OTC)
  - Chief Architect - SportingBet

The Problem

The Problem

- **Change is the main constant of software development**
- **We often struggle with change**
  - Leads to production problems
- **Why do we struggle?**
- **Can we do better?**

What Can Change?

- **Infrastructure**
  - Move to new provider
  - System upgrade
- **Users**
  - Sudden growth
  - Continual, steady growth
- **Code**
  - New release
  - Library upgrades
  - New subsystems

The Problem

"No matter what they tell you, it's always a people problem."

- Gerald Weinberg

Two Different, Related Problems

- **Outages**
  - Complete loss of service to customers
  - Especially caused by altering PROD environment
  - Post-release outages

- **Performance Problems**
  - Degraded, but may not be completely unavailable
  - Can occur pre-release or post-release
  - Often caused by inadequate performance testing

What Is Performance?

- **Measurement-driven approach to understand application behaviour under load**
  - Important: Measurement-driven
- **This sets up a battle between people & data**
- **Performance is a huge topic**
  - Let's choose our special focus

Goals & Objectives

- **Reduce Risk of Change**
  - Operational, Delivery & Reputational
- **Improve reliability**
  - Reduce cost from outages
- **Increase maintainability**
  - Reduce overall platform cost
- **Quantifiable impact**
  - Ideally in $$$ saved

Study The Problems

- How can learn how to do better?
- Study the problem
- Look at failure cases
- Learn from them
- See how to avoid them

Example Problems (Antipatterns)

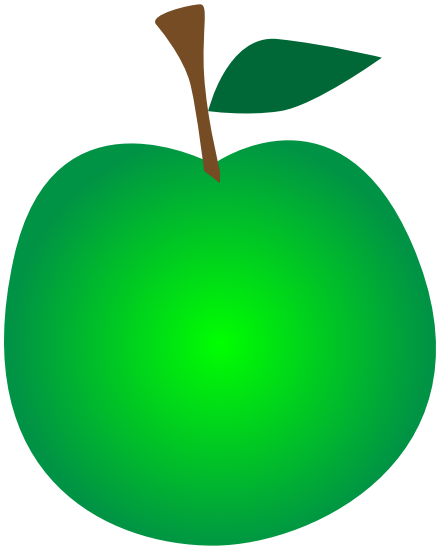
- **Misunderstood Production Environment**
  - UAT is My Desktop
  - PROD-like Data is Hard
- **Configuration Problems**
  - Play With Environment Switches
  - Configuration By Fairy Tales
- **Inappropriate Focus**
  - Blame Donkey
  - Micro-analysis
- **Silo Mentality**
  - Throw It Over The Wall

UAT is my desktop

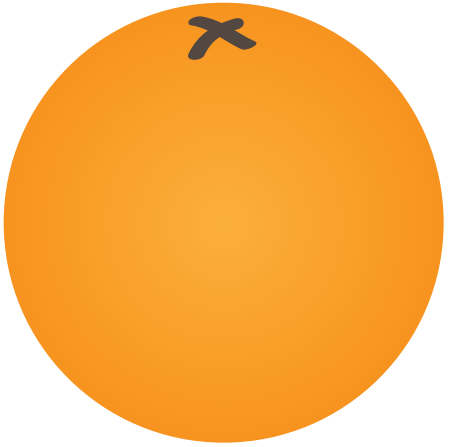
UAT is my desktop

- **UAT environment differs significantly from PROD**
  - "A full-size UAT environment would be too expensive"
- **Reality**
  - Track cost of outages & lost customers
  - Outages caused by differences in environments are almost always more expensive than a few more boxes
- **Modern software has complex runtime behaviour**
  - Especially true of adaptive environments (JVM, CLR)
  - “Some unrepresentative UAT is better than nothing” is a dangerous half-truth

PROD-like Data is Hard

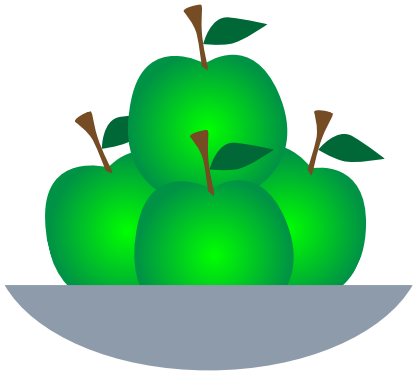


Testing Data

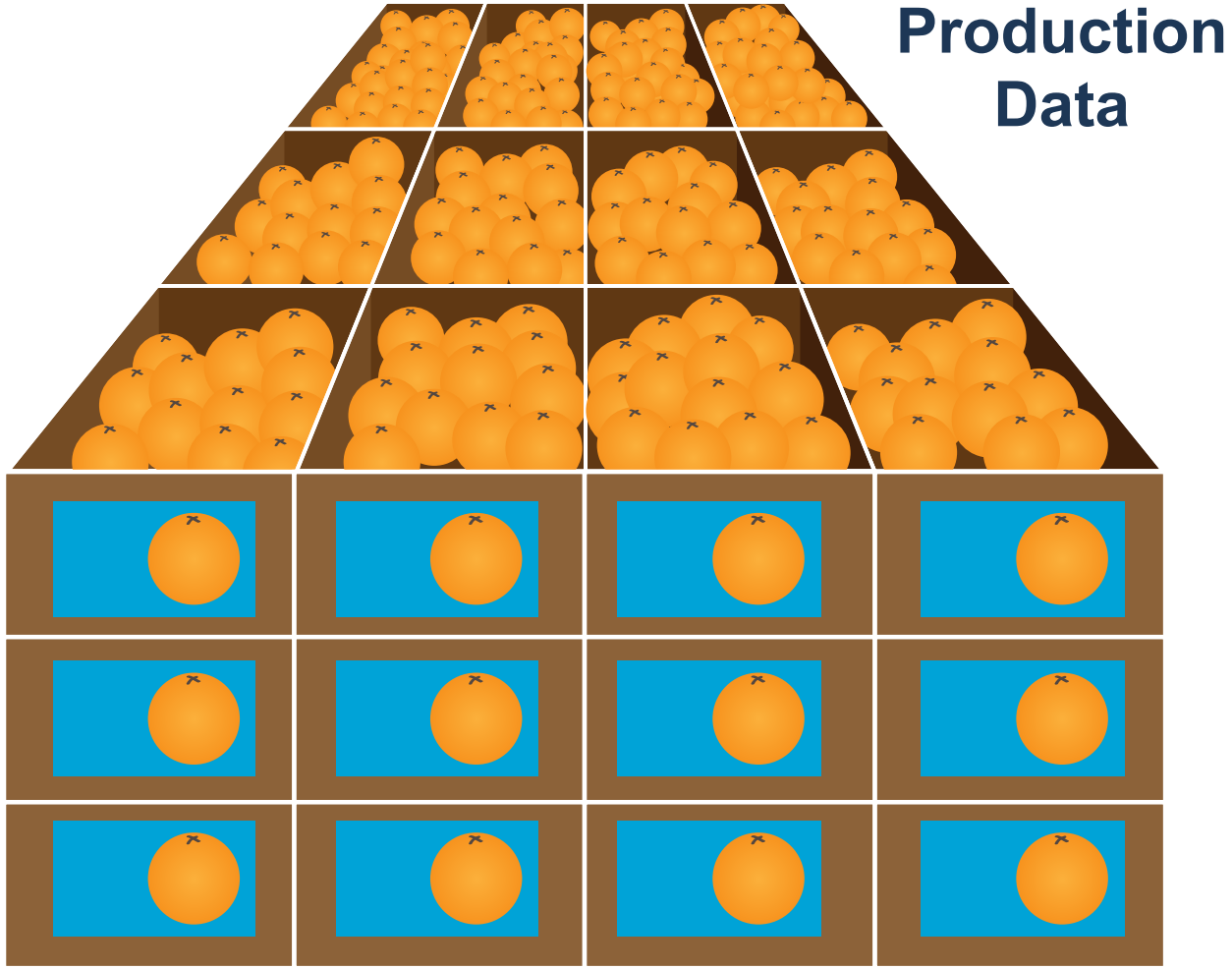


Production Data

PROD-like Data is Hard



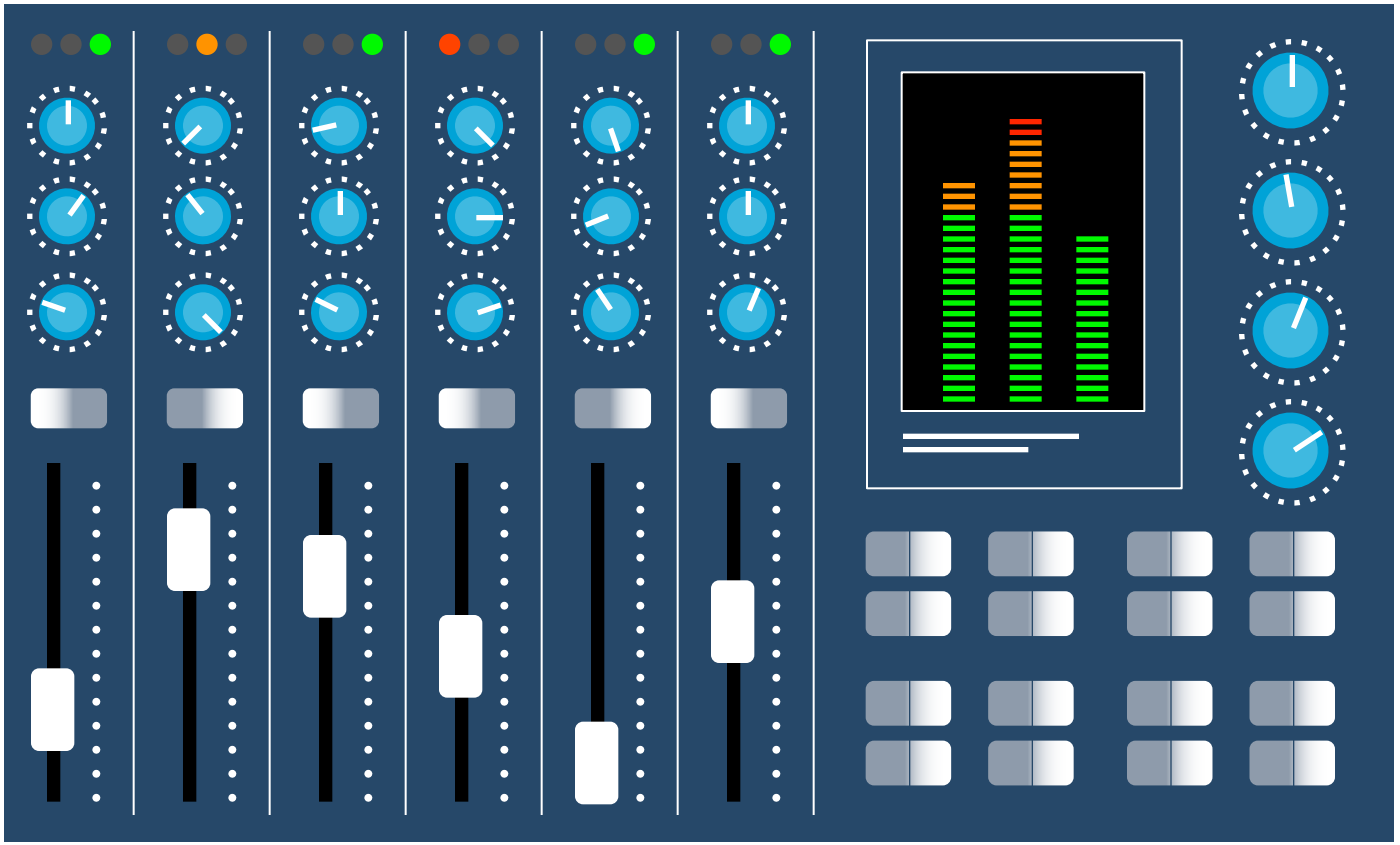
Testing Data



PROD-like Data is Hard

- **Data in UAT looks nothing like PROD**
  - "It's too hard to keep PROD and UAT data the same"
- **Reality**
  - Data in UAT must be PROD-like for accurate results
- **Another example of: “something is better than nothing”**
  - Even less true with data than environment setup
  - Can promote a false sense of security
- **Solutions**
  - Invest in formal data refresh process PROD -> UAT
  - Over-prepare for launching at big scale

Play With Switches

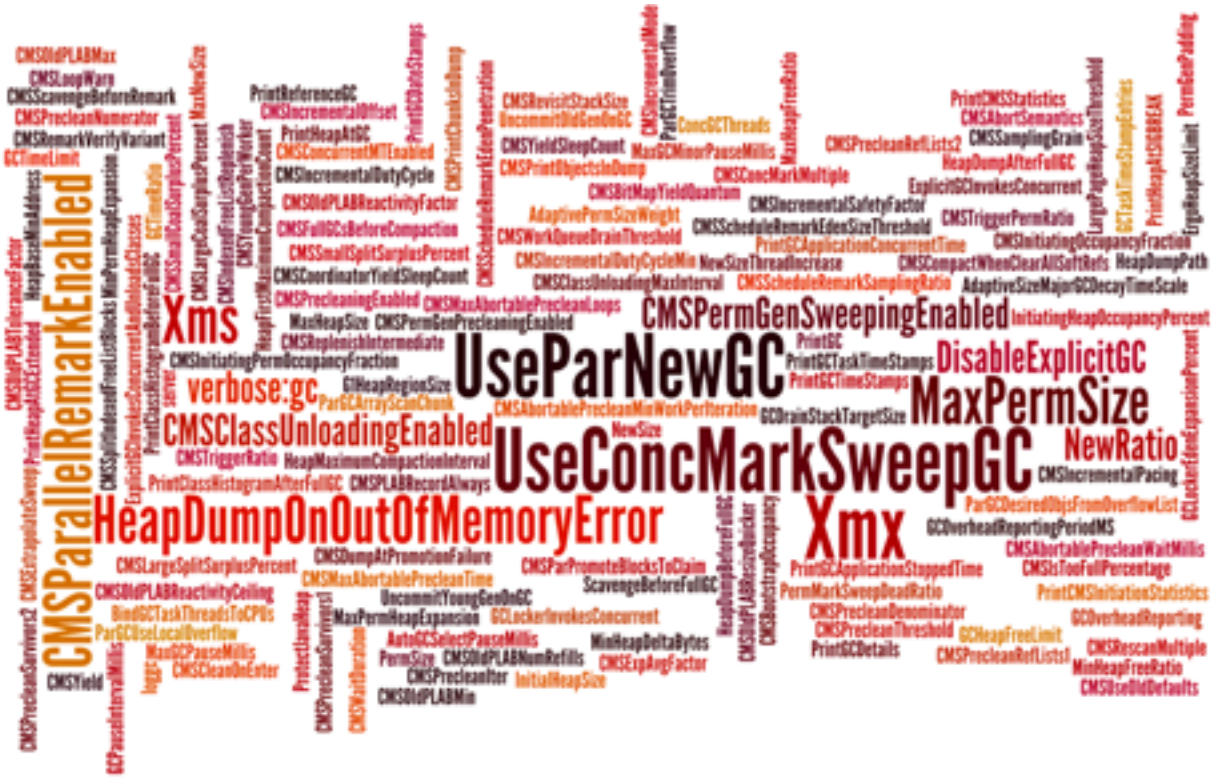


Play With Switches

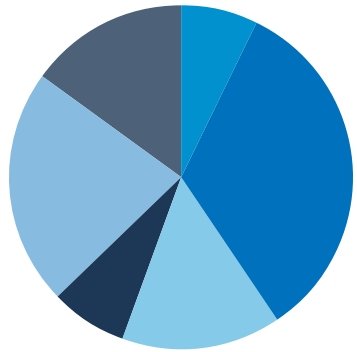
- **Team becomes obsessed with application switches**
  - "If I just change these settings, we'll get better results"
- **Reality**
  - Team does not understand impact of changes
- **Before putting any change to switches live**
  - Measure in PROD
  - Change 1 switch at a time in UAT
  - Test change in UAT
  - Retest in UAT
  - Have someone recheck your reasoning
  - Only then PROD
  - Retest in PROD

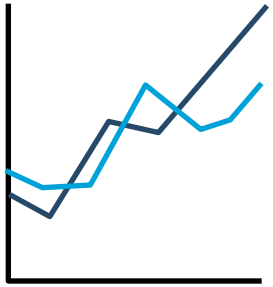
Case Study: Java Virtual Machine

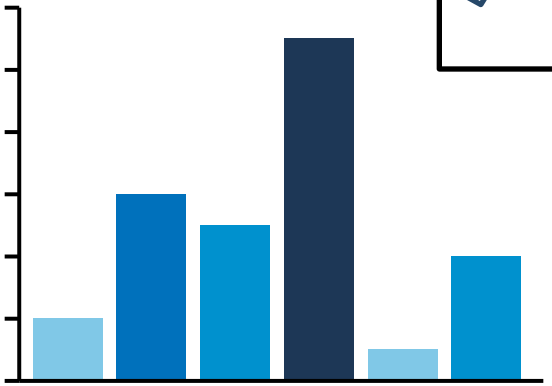
- **JVM has <u>hundreds</u> of switches**
  - Highly configurable runtime
  - Great temptation to play - RESIST
  - Defaults & self-management often sufficient



Configuration by Fairy Tales







NOT

Configuration by Fairy Tales

- **Code and switches changes are being applied blind**
  - "I found these great tips on Stack Overflow / Baidu / Google"
- **Reality**
  - Developer does not understand the context or basis of tip
  - True impact is unknown
  - A tip is a workaround for a known problem
  - Admin manuals contain general advice devoid of context
- **Solutions**
  - Only apply well-tested & well-understood techniques
  - Change directly affect the most important aspects of system

Performance Tuning is NOT

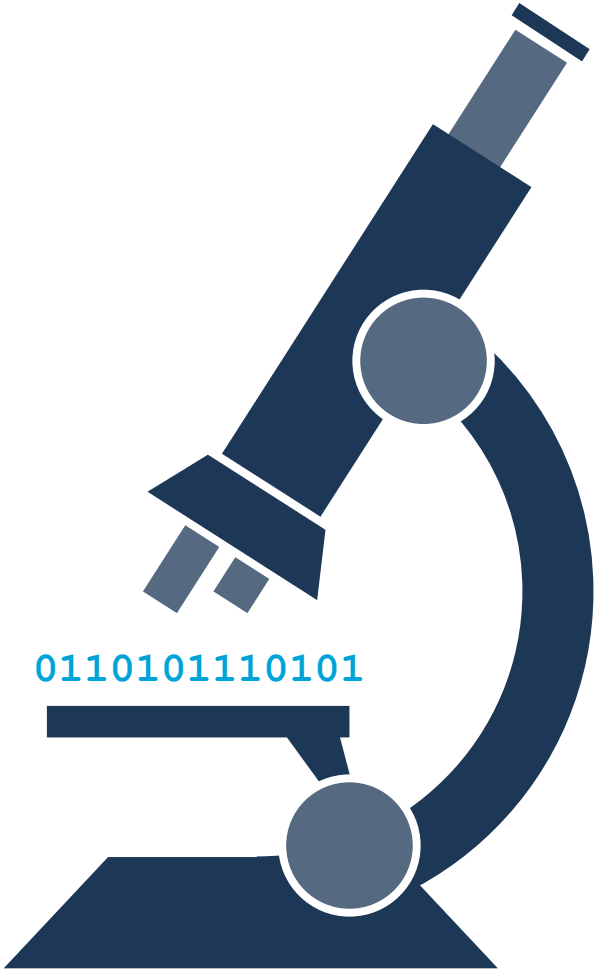
- A collection of tips and tricks
- Secret sauce
- Magic dust that you sprinkle on at the end of a project
- The province of “heroes”
- Particularly intellectually demanding

Blame Donkey

Blame Donkey

- **Certain components are always identified as the issue**
  - "It's always JMS / Hibernate / ANOTHERTECH"
    - AntiPattern often displayed by management / biz
    - But Technology is not immune
- **Reality**
  - Insufficient analysis has been done to reach this conclusion
- **Solutions**
  - Resist pressure to rush to conclusions
  - Perform analysis as normal
  - Communicate the results of analysis to all stakeholders

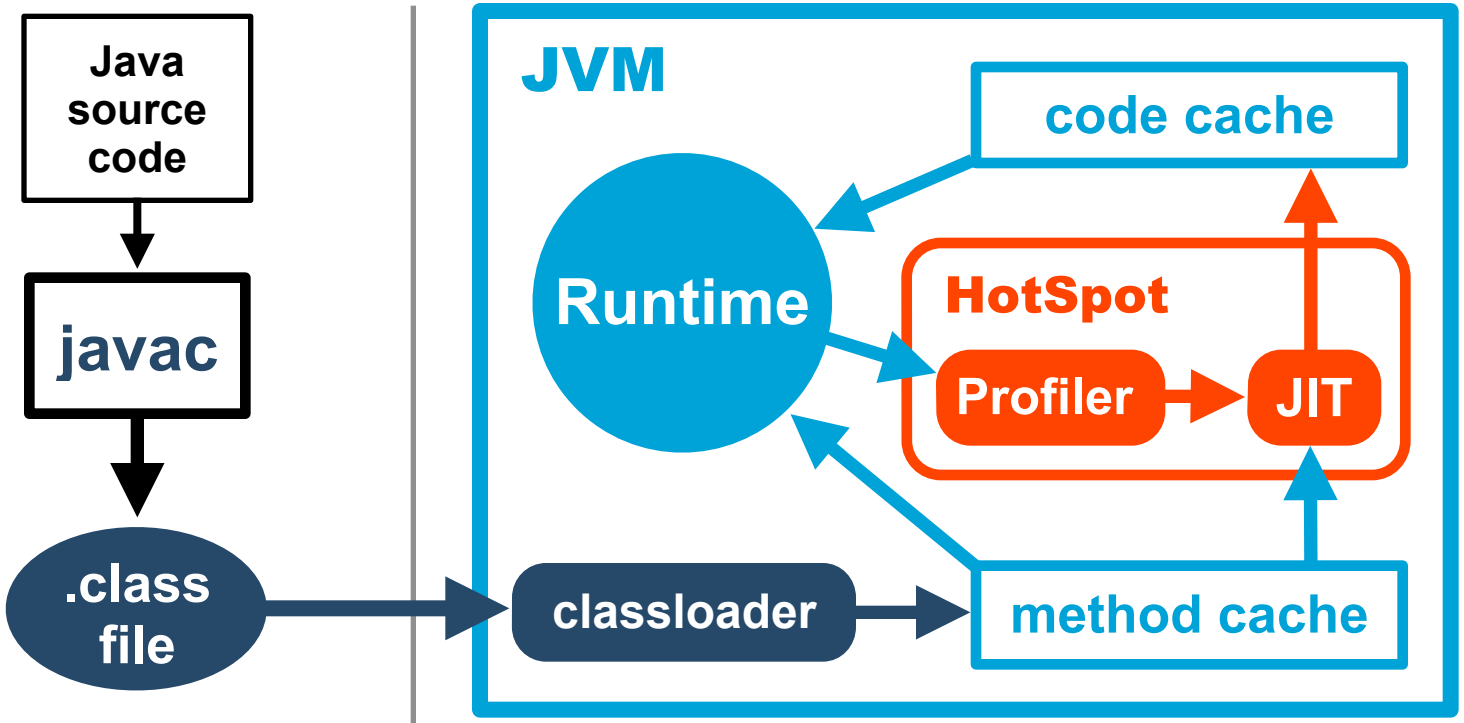
Micro-analysis



Micro-analysis

- **Effort focused on very low-level aspects of the system**
  - "If we can just speed up method dispatch time..."
- **Reality**
  - Overall impact of micro-changes is unknown
  - Can actually harm system performance
- **Solutions**
  - Do not micro-analyse
  - Unless your project is a known use case for it
    - E.g. Building a general purpose library or framework
    - Know the tradeoffs
  - Use existing tools for the analysis
    - Don't build your own analysis tool
    - Always discuss your results widely

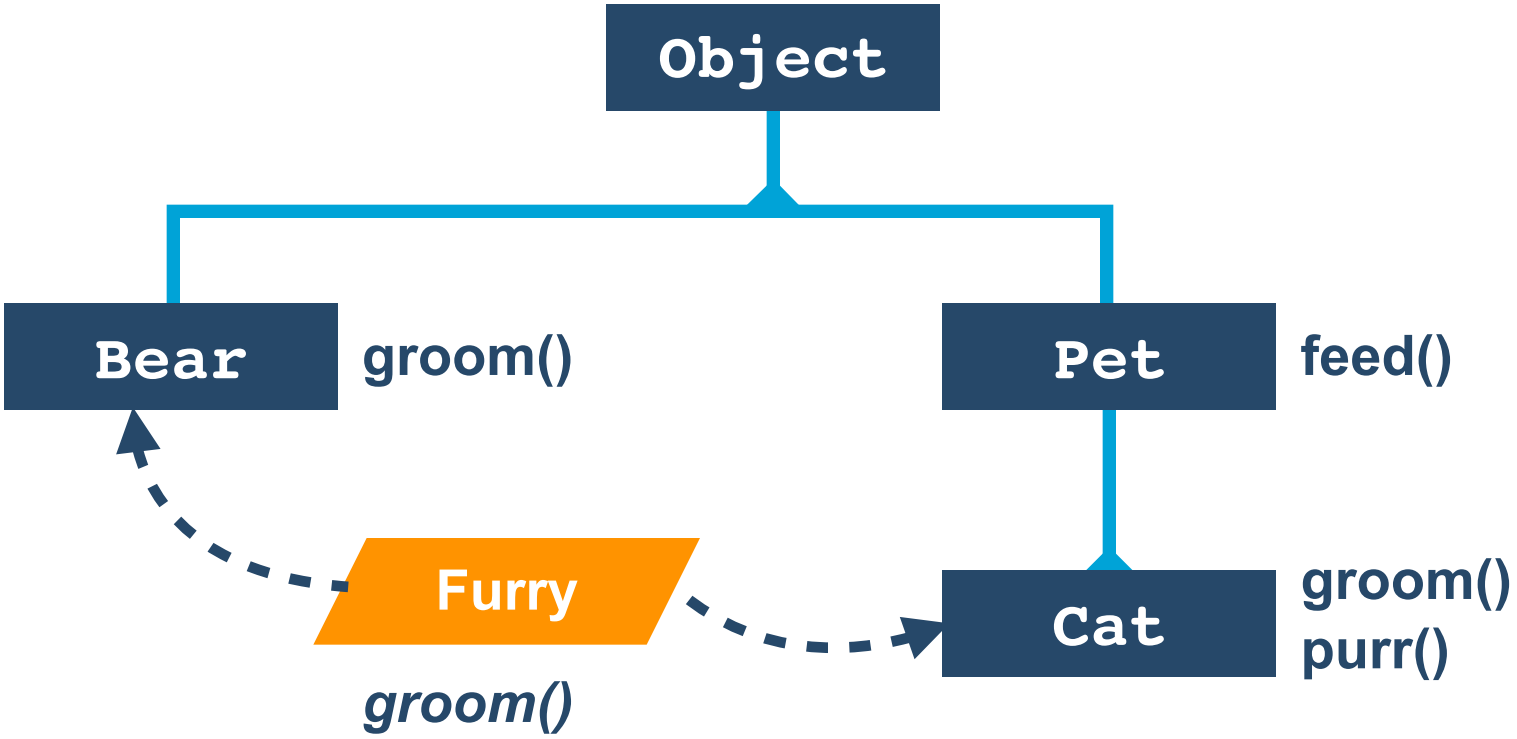
Example: What a modern JVM looks like



Class file creation

Continuous (Just In Time) compilation

Class Hierarchy



Example Code

- **Let's look at how we might use these classes, in a simple example:**

```
Bear baloo = new Bear();
Cat tiddles = new Cat("Tiddles");

tiddles.feed();
tiddles.groom();
baloo.groom();

Furry b = baloo;
b.groom();
```

Example Output

- **Here's the example output:**

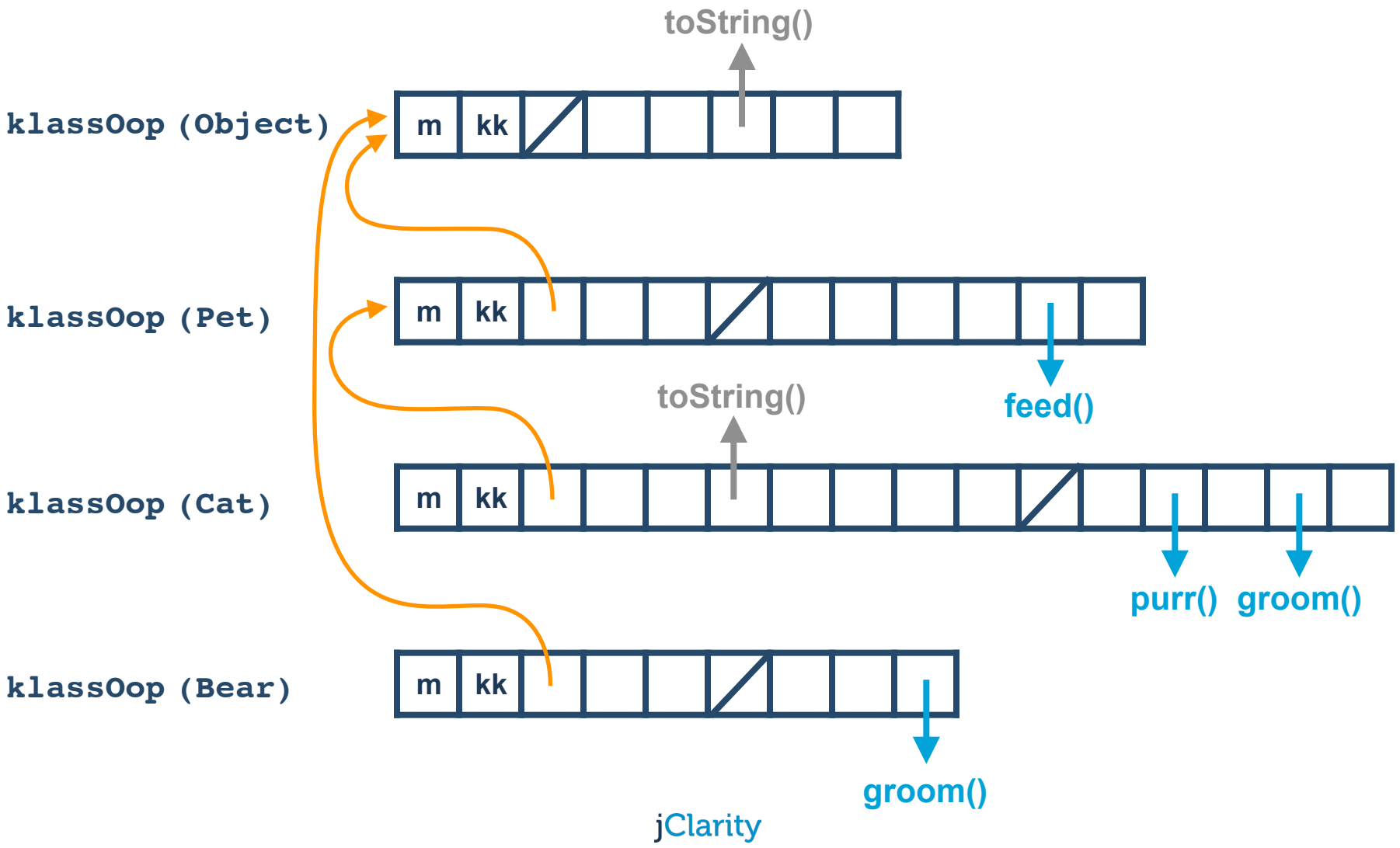
```
Tiddles has been fed
Tiddles washes itself.
The bear combs its fur with its claws. Don't get too close.
The bear combs its fur with its claws. Don't get too close.
```

Bytecode

- **Disassemble this using javap:**

```
Code:
   0: new           #5                  // class bootsandcats/Bear
   3: dup
   4: invokespecial #6                  // Method bootsandcats/Bear."<init>":()V
   7: astore_1
   8: new           #7                  // class bootsandcats/Cat
  11: dup
  12: ldc           #8                  // String Tiddles
  14: invokespecial #9                  // Method bootsandcats/Cat."<init>":(Ljava/lang/String;)V
  17: astore_2
  18: aload_2
  19: invokevirtual #10                 // Method bootsandcats/Cat.feed:()V
  22: aload_2
  23: invokevirtual #11                 // Method bootsandcats/Cat.groom:()V
  26: aload_1
  27: invokevirtual #12                 // Method bootsandcats/Bear.groom:()V
  30: aload_1
  31: astore_3
  32: aload_3
  33: invokeinterface #13,  1           // InterfaceMethod bootsandcats/Furry.groom:()V
  38: return
```

VTable Layout



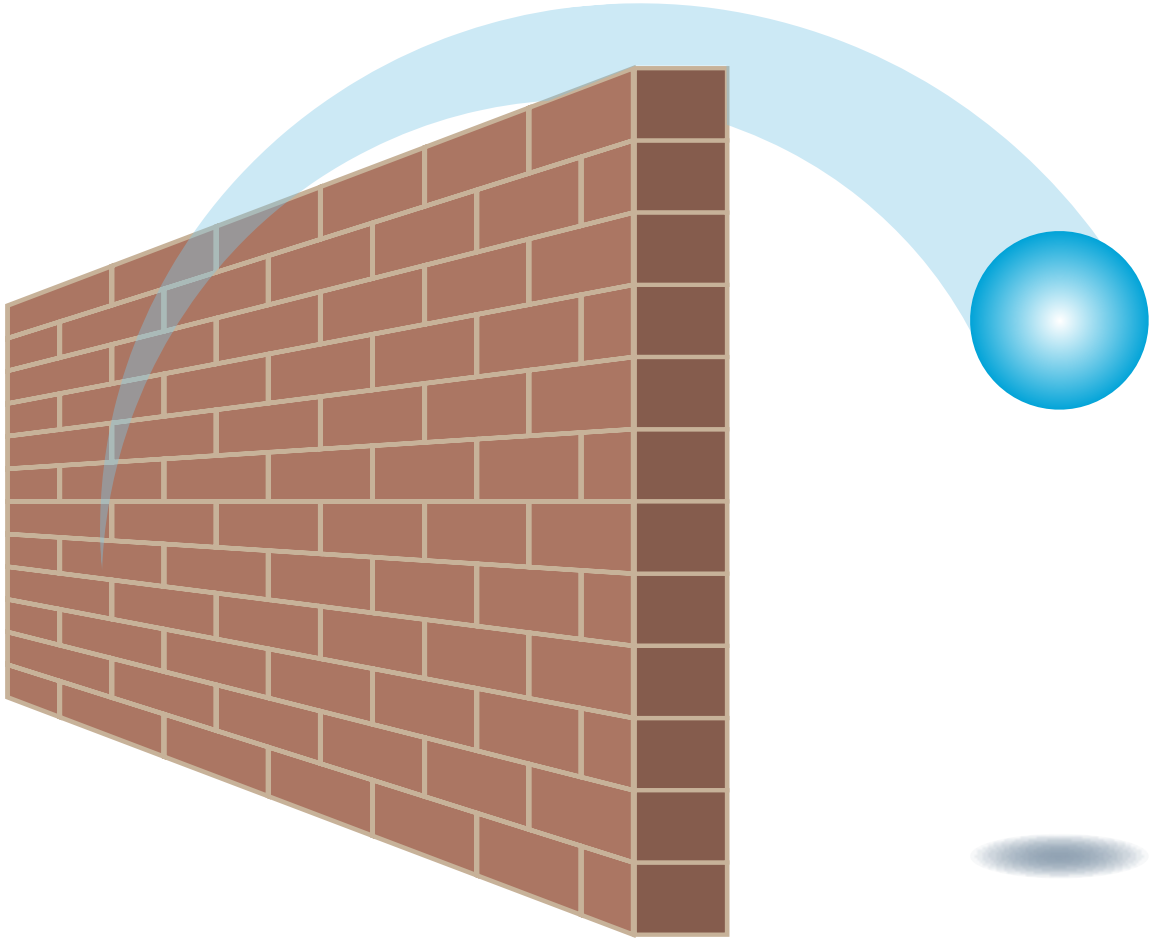
So What…?

Moral of the Story

- **Always Measure**
  - Always ensure that the measurement is telling what you think it is…

- **“The first principle is that you must not fool yourself, and you are the easiest person to fool.”**
  - Richard Feynman

Throw It Over The Wall



Throw It Over The Wall

- **Teams focus on their immediate domain**
  - "It has plenty of unit tests. QA should take over from here"
- **Reality**
  - Preventable problems pass into production
- **Building better cross-team relationships helps**
  - Seek to understand what other teams actually do
  - Outages and releases are much less painful
  - Hard to do with remote L1 operations teams

Recurring themes in the antipatterns

- **Communication Problems**
  - Silo Mentality
- **Cognitive Biases**
  - Blame Donkey
  - Micro-analysis
- **Complex, adaptive software**
  - Uneven distribution of knowledge in dev team
  - Micro-analysis
- **Need to address these recurring themes**

Understanding Architecture

- Can you draw the architecture of your whole app from memory?

Understanding Architecture

- **Diagrams!**
  - Component Diagrams
  - Sequence Diagrams
  - Data Flow Diagrams
- **Domain Language**
  - Naming matters a lot
- **Deployment**
  - How is the application laid out across machines / VMs
- **If you don't understand your architecture, how can you hope to cope in an outage?**

Addressing the antipatterns with data

- **Collect monitoring data**
  - As a formal process, not ad-hoc
  - Data should be retained & centralized
- **Analyse data**
  - Often the missing link
  - Data analysis & interpretation skills may be scarce
  - Where possible, use tooling to autogenerate reports
- **Ensure all teams see the output reports**
  - Communication between teams is essential
- **Understand what normal system function looks like**

Measurement & Statistics

- **Best tool against cognitive biases is data**
- **Need logging & monitoring**
  - But also analysis
  - Data can overwhelm
  - Patterns aren't always easy to spot by eye
- **Proper collection processes are needed**
  - Too many outages are analysed via ad-hoc data
- **Ensure sufficient logging**
  - Can we retrace all the steps of an outage?

Why Measure?

- **Humans are poor at guessing**
- **We have cognitive biases**
  - Especially Confirmation Bias
- **Developers tend to think along "golden paths" in code**
  - Testers are trained to think down darker paths
- **Modern systems are exceedingly complex**
  - Lots of external factors
    - Virus scans, other apps, backups

Performance Distilled

- **There is no such thing as perfect performance**
  - Only acceptable
- **End users define your performance requirements**
- **Performance is a non-functional requirement**
  - Should be specified like any other NFR
  - Must have quantitative goals related to system observables
  - Should be set in conjunction with system stakeholders

Performance Testing

- **Performance Tuning is an iterative process**
  - Identify worst problem by measurement & analysis
  - Reconfigure or refactor to remove it
  - Measure again. Performance acceptable now?
  - If not, iterate

- **Great example of cross-functional activity**
  - Needs PROD-like UAT
  - Needs realistic data set
  - Needs proper configuration
  - Needs representation from all technical teams

Evidence-Based Operations

- **Ask Key Questions**
- **Define Key Quantities & Observables**
  - Use to quantify questions wherever possible
- **State Assumptions & Approximations Explicitly**
- **Proceed Methodically**
- **Check Your Working**
  - Consult Your Peers

(Journalist's) Key Questions

- What happened…?
- Where did it happen…?
- When did it happen…?
- What / who was involved… ?
- Why do we think it happened… ?
- Do we think it will happen again… ?

Key Operational Questions

- What observables of your code are you measuring?
- How are you measuring those observables?
- What are the goals for the observables?
- How will you recognize when you're done?
- What is the acceptable cost for system tuning?
- What can't be sacrificed as you optimize & change things?

Explicit Assumptions & Approximations

- **State**
  - How much data? How many concurrent users?
- **Document**
  - Work with other stakeholders
  - Discuss with peers
  - Make formal document as a project artefact
- **Revisit**
  - Check every few months
  - Verify that initial assumptions are still true
  - Update if not
- **Remember: THINGS WILL CHANGE**

Defining Production Analysis

- **An measurement-driven approach to understanding an applications behaviour under load**
  - Measurements
  - Statistics
  - Data analysis
- **Experimental Science**
  - Has inputs & outputs
- **Answer questions such as:**
  - At 10x customers, will the system have enough memory to cope?
  - What is average response time customers see from the application?
  - What does the rest of that distribution look like?
  - How does that compare to your competitors?

Real Science is Repeatable

- **Check your working**
- **Verify your results**
- **Provide information for your peers to replicate your results**
  - Do NOT just reuse others conclusions blindly

"After you've not fooled yourself, it's easy not to fool other scientists." - Richard Feynman

Final Thought

- **Communication**
  - Learn to understand how other teams actually work
  - Share tools across teams
- **Data Collection & Monitoring**
  - Standardise on monitoring & data collection
  - Check that apps log sufficient & appropriate detail
  - Beware of bit rot & change from releases
- **Analysis & Reporting**
  - Data needs to be analysed & reported to teams
- **Result: Reduced risks & better decisions**

THANK YOU

Commercial - www.jclarity.com @jclarity

Products

jClarity Censum: The world's best GC log analysis tool

jClarity Illuminate: The smart, learning performance problem finder

Community - www.meetup.com/londonjavacommunity

Ben Evans - ben@jclarity.com

BOOK SIGNING TODAY

BOOK SIGNING

TODAY 1310 AT TURING BOOK STAND

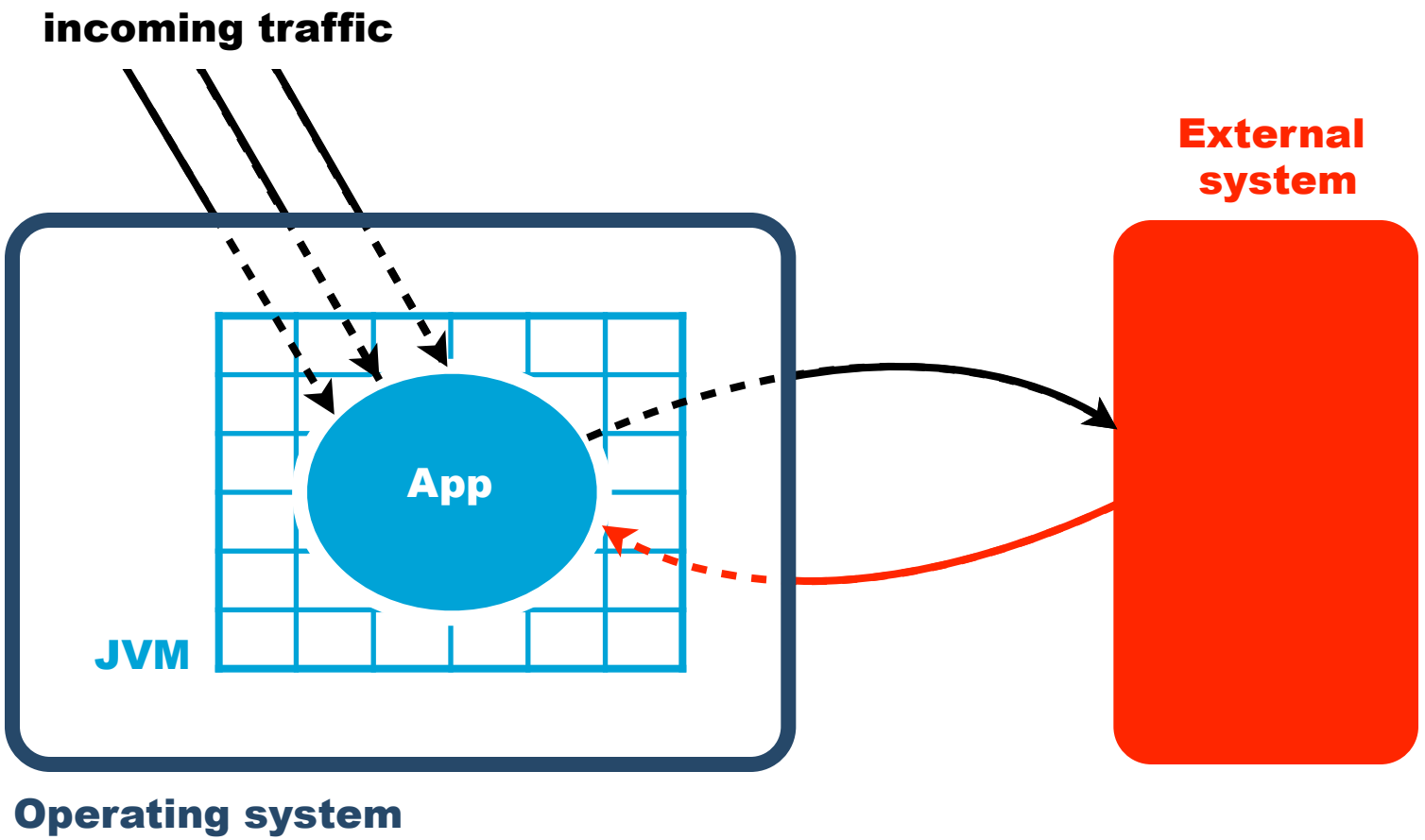
Java程序员修炼之道

NEW BOOK (2015)

Java权威技术手册（第6版）

预计出版日期：**2015年6月**

External Systems



External Systems

- **Use a thread profiler to identify waiting threads**
  - Then identify what these threads are waiting on
- **Some Common Problems**
  - A database query not performing well
  - Poor network latency
  - Slow SOAP / REST call not performing well
  - Poor throughput from an RMI service
  - Authentication / Authorization System overwhelmed

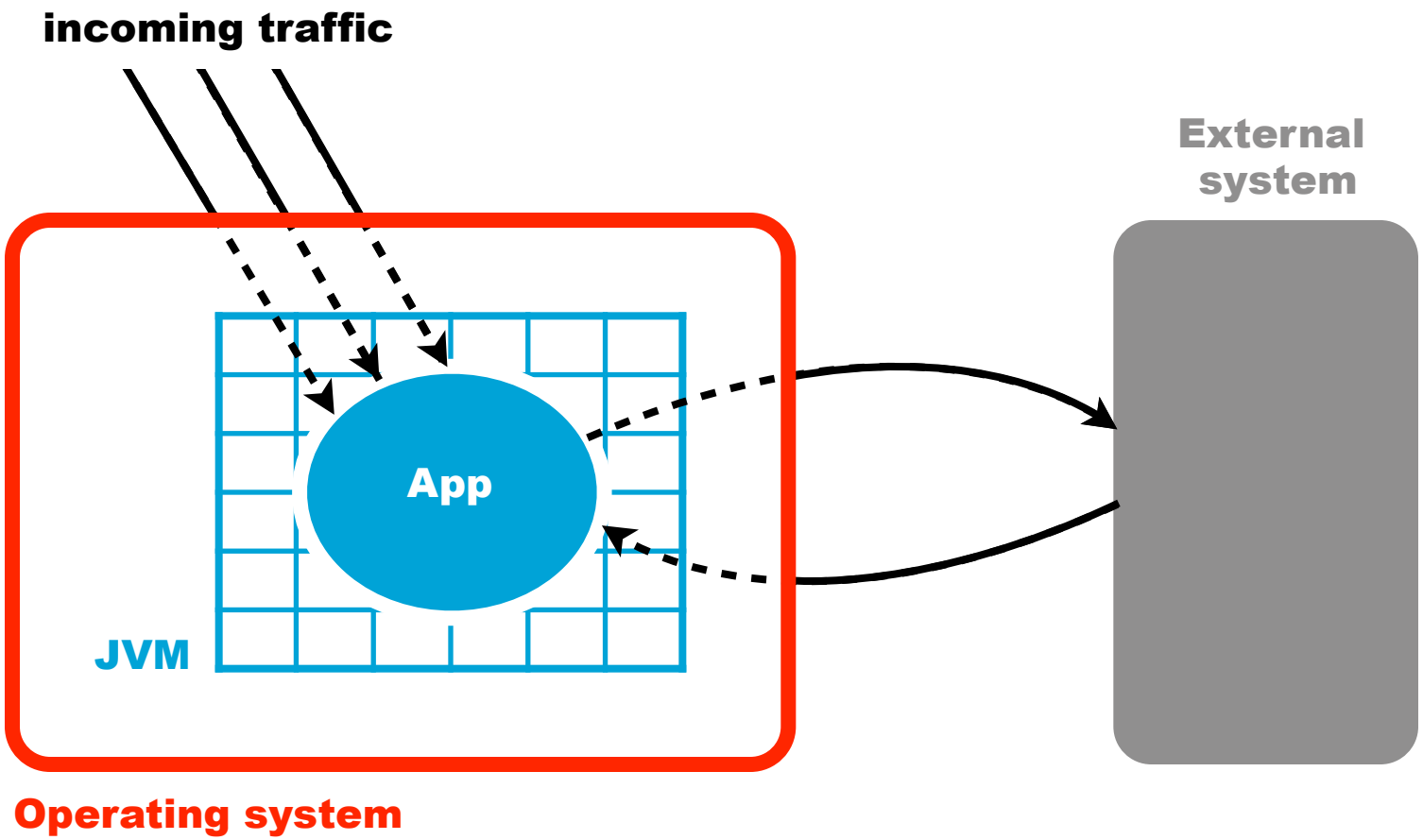
External Systems

- **To fix, apply a code or architectural change**

Examples

- **Improve the throughput of the external resource**
  - Spin up more database servers
- **Communicate with the external system less often**
  - By caching the results of previous calls
- **Interact with the external system in a more efficient manner**
  - Batching updates

Operating System



Operating System

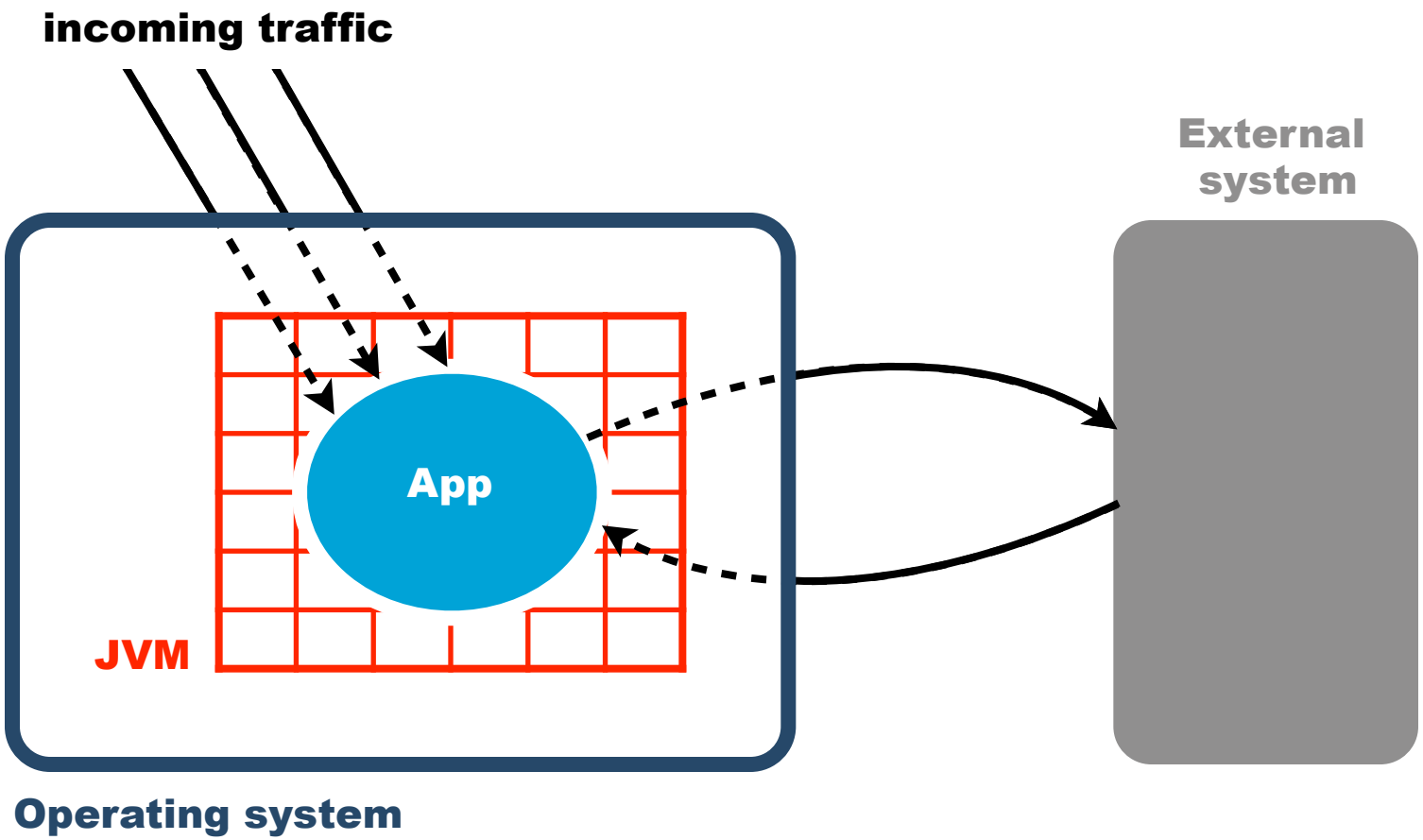
- **The OS can manifest a number of performance problems**
  - Disk Read
  - Disk Write
  - Context Switching
- **Disk Problems**
  - Slow hardware
  - Inefficient Use
- **Context Switching**
  - OS spending too much time switching between tasks
  - Threads contending for software lock
  - Contention for hardware resources

Operating System

- **Disk Problems**
  - Get SSDs (if you haven't already)
  - Use an in-memory cache (for read data)
  - Batch writes
  - Investigate overall usage profile
    - Are there large spikes at particular times?

- **Context Switching**
  - Move contending processes
  - Investigate locking strategies

JVM / GC


incoming traffic
External system
App
JVM
Operating system

JVM / GC

- **The usual JVM-based cause of performance issues is GC**
  - Occasionally other subsystems can cause bottlenecks
- **Solving these issues & tuning GC is non-trivial**
  - Tooling can definitely help
- **Use an up to date JVM**
  - Oracle JDK 6 is EOL & a **critical** PROD risk
  - JDK 7 is 20-30% faster than 6 (& JDK is faster again)
  - JDK 7 & 8 have features with major productivity increases

Why GC Log Files?

- **Log files can be processed after the fact**
  - GC Log files contain a lot of information
  - Simple mechanism to collect logs
  - Often critical information in an outage
- **Every JVM in PROD should log GC events**
  - At least the mandatory flags
  - The GC log is a great source of information
  - Especially useful for “cold case” analysis
- **Runtime MXBeans**
  - Need custom mechanism to automatically collect
  - Intended for point-in-time use
  - Impact the running application
    - Can cause their own GC problems

GC Logging - Mandatory Flags

`-Xloggc:<pathtofile>`

Path to the log output, make sure you've got disk space!

`-XX:+PrintGCDetails`

Minimum information for tools to help

Replace `-verbose:gc` with this

`-XX:+PrintTenuringDistribution`

Premature promotion information

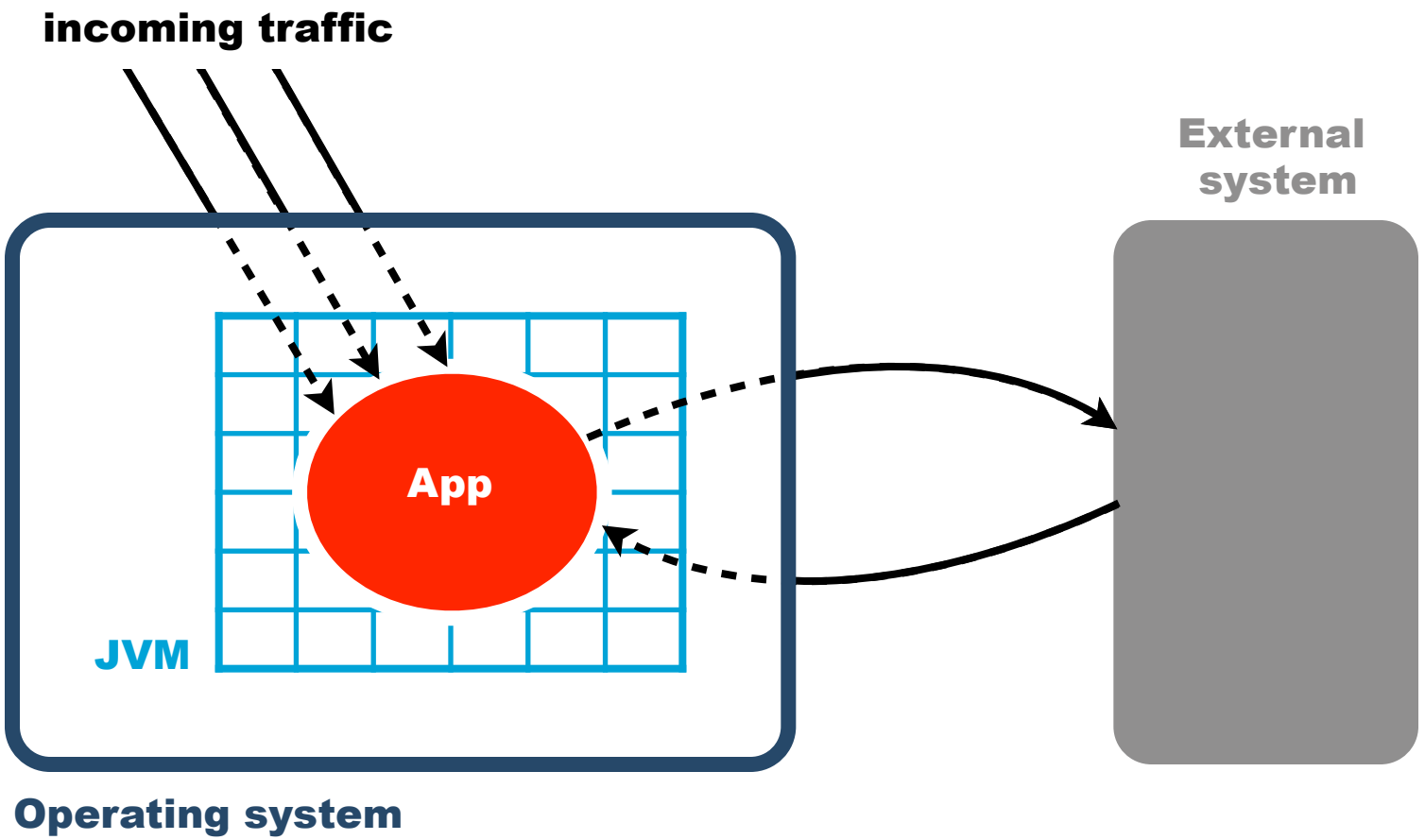
`-XX:+PrintGCTimestamps`

Capture the timestamp of GC events

Tooling

- **HP JMeter (Google it)**
  - Free, reasonably solid, but no longer supported / enhanced
- **GCViewer (http://www.tagtraum.com/gcviewer.html)**
  - Free, OSS, but ugly & somewhat limited
- **GarbageCat (http://code.google.com/a/eclipselabs.org/p/garbagecat/)**
  - Best name, but not very active project
- **IBM GCMV (http://www.ibm.com/developerworks/java/jdk/tools/gcmv/)**
  - Only choice for IBM J9 support
- **jClarity Censum (http://www.jclarity.com/products/censum)**
  - Best in class

Application Code



Application Code

- **Apply a code profiler**
  - VisualVM, hprof, JProfiler
- **How commonly is this the cause of performance problems?**

Application Code

- **Apply a code profiler**
  - VisualVM, hprof, JProfiler
- **How commonly is this the cause of performance problems?**
- **Usually the rarest of performance problems**
  - Regardless of what developers think
- **Anecdotal experience**
  - The true cause of slowness <10% of the time
- **Don’t just jump in with a profiler**

Too Much Traffic